Data Handling using DSA

CEP Final Report

By

NAME	RegistrationNumber
Hamza Jadoon	CUI/FA22-BCE-012/ATD

For the course

Data Structures and Algorithm

Semester Spring

2024

Supervised by:

Sir Bicktash Ali Jehangir

Department of Electrical & Computer Engineering

COMSATS University Islamabad – Abbottabad Campus

Table of Contents

Introduction **3**

Objectives **3**

Methodology **3**

Purpose of Making this Project **4**

Advantages of Making this Project **4**

Project Complexity **5**

Code Implementation: **6**

Introduction

In this project, we implemented a program to manage data records consisting of a Name, Contact, and Address. The program supports operations such as insertion, searching, retention in secondary memory, and loading from a data file on disk. We utilized four different data structures for the implementation: Singly Linked List, Doubly Linked List, Stack, and Queue. The primary objective was to compare the performance parameters of these data structures in handling the specified operations.

Objectives

- Implement data management using Singly Linked List, Doubly Linked List, Stack, and Queue.
- Compare the performance of these data structures in terms of insertion, search, and file I/O operations.
- Retain data in secondary memory and load it back into the data structures.

Methodology

1. Singly Linked List (Insertion at End)
    - Insertion: Adding nodes at the end of the list.
    - Search: Traversing the list to find a specific record.
    - Retention: Saving the list to a file.
    - Loading: Reading the list from a file.

2. Doubly Linked List (Insertion at Beginning)
    - Insertion: Adding nodes at the beginning of the list.
    - Search: Traversing the list to find a specific record.
    - Retention: Saving the list to a file.
    - Loading: Reading the list from a file.

3. **Stack**
    - **Insertion**: Pushing nodes onto the stack.
    - **Search**: Traversing the stack to find a specific record.
    - **Retention**: Saving the stack to a file.
    - **Loading**: Reading the stack from a file.

4. **Queue (Circular Queue)**
    - **Insertion**: Enqueuing nodes into the queue.
    - **Search**: Traversing the queue to find a specific record.
    - **Retention**: Saving the queue to a file.
    - **Loading**: Reading the queue from a file.

Each record consists of:

- Name: Up to 50 characters.
- Contact: Up to 15 characters.
- Address: Up to 100 characters.

Purpose of Making this Project

The primary purpose of this project is to explore and compare the efficiency and performance of different data structures in managing data records. By implementing the same operations across multiple data structures, we can gain insights into their strengths and weaknesses, which will help in selecting the appropriate data structure for specific use cases in real-world applications.

Advantages of Making this Project

- **Educational Value**:

  Enhances understanding of fundamental data structures and their operations.

- **Performance Analysis**:

Provides empirical data on the efficiency of various data structures in handling common operations.

- **Practical Application**:

  Demonstrates the practical use of data structures in managing real-world data records.

- **Skill Development**:

  Improves programming and problem-solving skills in C language and data management.

Project Complexity

The project involves multiple aspects of data management and requires a good understanding of data structures, file handling, and memory management in C. Each data structure has its own complexity:

- **Singly Linked List**:

  Moderate complexity due to dynamic memory allocation and pointer manipulation.

- **Doubly Linked List**:

  Higher complexity compared to singly linked list due to the additional backward pointers.

- **Stack**:

  Relatively simple but requires careful management of push and pop operations.

- **Queue**:

  Complexity depends on the type of queue (circular queue in this case) and the handling of front and rear pointers.

Code Implementation:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NAME_SIZE 50
#define CONTACT_SIZE 20
#define ADDRESS_SIZE 100

typedef struct Data {
    char name[NAME_SIZE];
    char contact[CONTACT_SIZE];
    char address[ADDRESS_SIZE];
} Data;
 typedef struct SLLNode {
    Data data;
    struct SLLNode *next;
} SLLNode;
 typedef struct DLLNode {
    Data data;
    struct DLLNode *prev;
    struct DLLNode *next;
} DLLNode;
 typedef struct StackNode {
```

```
    Data data;
    struct StackNode *next;
} StackNode;
 typedef struct QueueNode {
    Data data;
    struct QueueNode *next;
} QueueNode;
typedef struct Queue {
    QueueNode *front;
    QueueNode *rear;
} Queue;
 void insertSLL(SLLNode **head, Data data) {
    SLLNode *newNode = (SLLNode *)malloc(sizeof(SLLNode));
    newNode->data = data;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    SLLNode *temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}
 void insertDLL(DLLNode **head, Data data) {
```

```
    DLLNode *newNode = (DLLNode *)malloc(sizeof(DLLNode));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = *head;
    if (*head != NULL) {
        (*head)->prev = newNode;
    }
    *head = newNode;
}
 void push(StackNode **top, Data data) {
    StackNode *newNode = (StackNode *)malloc(sizeof(StackNode));
    newNode->data = data;
    newNode->next = *top;
    *top = newNode;
}
 void enqueue(Queue *queue, Data data) {
    QueueNode *newNode = (QueueNode *)malloc(sizeof(QueueNode));
    newNode->data = data;
    newNode->next = NULL;
    if (queue->rear == NULL) {
        queue->front = queue->rear = newNode;
        return;
    }
    queue->rear->next = newNode;
    queue->rear = newNode;
}
```

```
void saveToFile(const char *filename, Data *data, int size) {
    FILE *file = fopen(filename, "w");
    if (file == NULL) {
        perror("Failed to open file");
        return;
    }
    for (int i = 0; i < size; i++) {
        fprintf(file, "%s,%s,%s\n", data[i].name, data[i].contact, data[i].address);
    }
    fclose(file);
}
 int loadFromFile(const char *filename, Data *data, int maxSize) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        perror("Failed to open file");
        return 0;
    }
    int count = 0;
    while (count < maxSize && fscanf(file, "%49[^,],%19[^,],%99[^\n]\n",
data[count].name, data[count].contact, data[count].address) == 3) {
        count++;

    fclose(file);
    return count;
}
 void generateRandomData(Data *data, int size) {
    for (int i = 0; i < size; i++) {
```

```
        snprintf(data[i].name, NAME_SIZE, "Name%d", i);
        snprintf(data[i].contact, CONTACT_SIZE, "Contact%d", i);
        snprintf(data[i].address, ADDRESS_SIZE, "Address%d", i);
    }
}
 void printData(Data data) {
    printf("Name: %s, Contact: %s, Address: %s\n", data.name, data.contact, data.address);
}
int main() {
    const int DATA_SIZE = 100;
    Data data[DATA_SIZE];
    generateRandomData(data, DATA_SIZE);
    SLLNode *sllHead = NULL;
    DLLNode *dllHead = NULL;
    StackNode *stackTop = NULL;
    Queue queue = {NULL, NULL};
    for (int i = 0; i < DATA_SIZE; i++) {
        insertSLL(&sllHead, data[i]);
        insertDLL(&dllHead, data[i]);
        push(&stackTop, data[i]);
        enqueue(&queue, data[i]);
    }
    saveToFile("data.txt", data, DATA_SIZE);
    Data loadedData[DATA_SIZE];
    int loadedSize = loadFromFile("data.txt", loadedData, DATA_SIZE);
```

```
    printf("Loaded Data from File:\n");
    for (int i = 0; i < loadedSize; i++) {
        printData(loadedData[i]);
    }
    return 0;
}
```

Resultant Output: